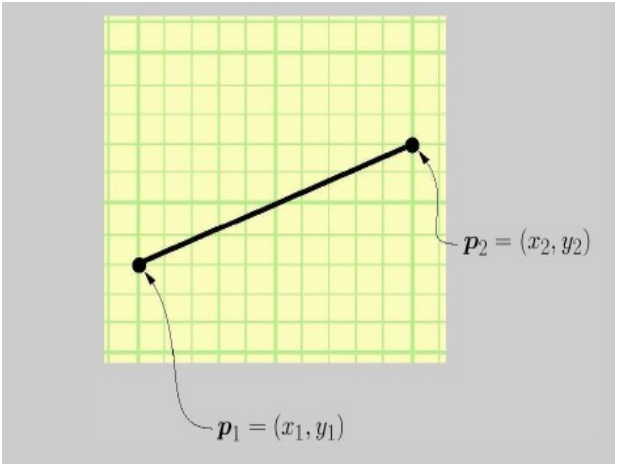
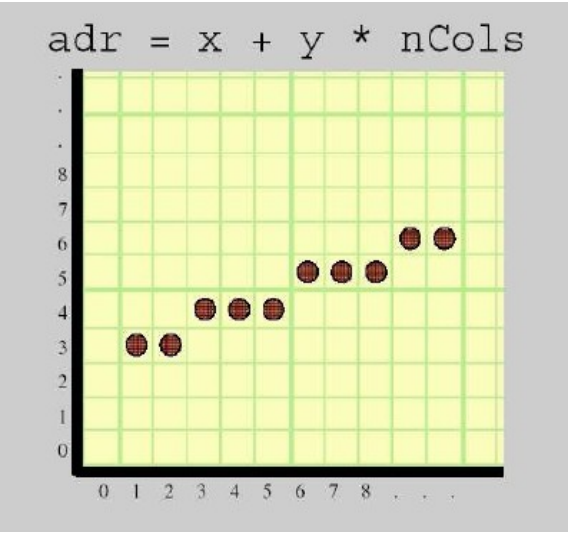
Digitalización (Scan Convertion)





Dpto. de Informática
Fac. Cs. Físico-Mat. y Nat.
Universidad Nacional De San Luis
Argentina

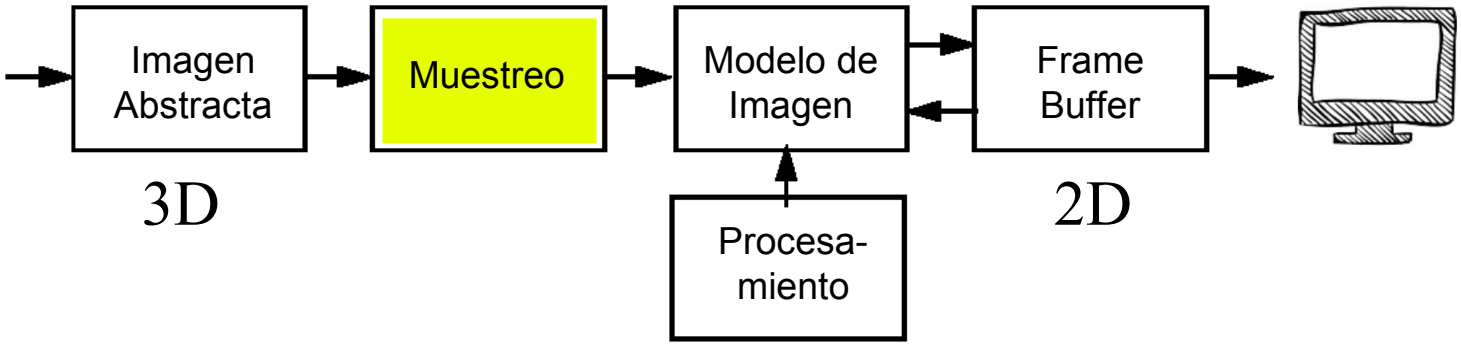
Computación Gráfica

- Según Fetter, CG referencia a la creación, manipulación y almacenamiento de modelos de objetos e imágenes.
- Simular un mundo 3D por medio de ***fórmulas matemáticas*** que describan la forma y el aspecto de los objetos.
  - Los modelos deben ser susceptibles de ser manipulados de forma efectiva por un computador
- Tres grandes aspectos a considerar:

  - Modelado de las Formas
  - Modelado de la Apariencia
  - Visualización

Rendering - Generación de la Imagen

- Es el proceso por medio del cual se obtiene una representación estática 2D de un mundo abstracto 3D.



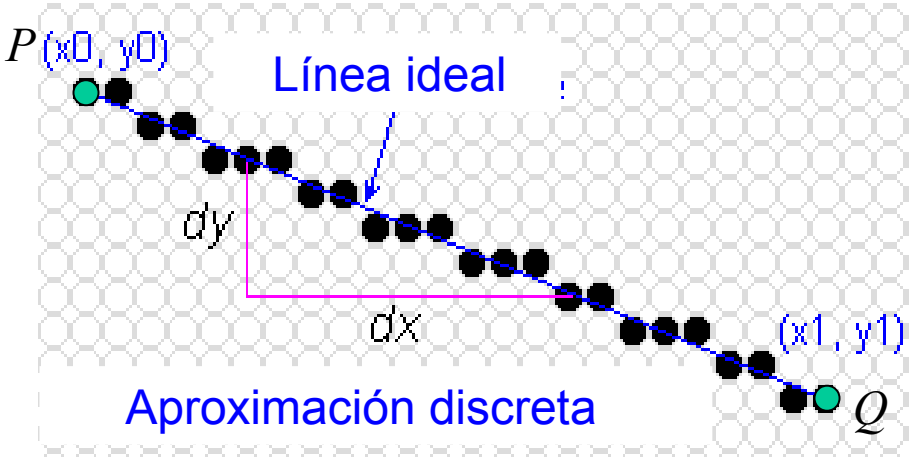
- Una imagen abstracta se encuentra definida y existe dentro de la computadora como una imagen continua.
- La imagen resultante, en el Frame Buffer es una imagen discreta.
- El **muestreo** es el proceso por el cual se convierte una imagen continua en discreta.
- El hardware de visualización de un sistema gráfico convierte los valores de intensidad discretos en voltajes analógicos continuos aplicados a un dispositivo específico.

Convertir Líneas en Puntos

(scan convertion)

Problema

- Dados 2 puntos P y Q en el plano, ambos con coordenadas enteras, determinar que pixels del dispositivo de visualización deben ser encendidos con el objeto de pintar un segmento de línea del ancho de un pixel que comience en P y termine en Q.



Dibujar una línea

- Problemas con el enunciado
  - No es claro el concepto de “dibujar”.
  - No establece cuales deben ser los pixels a ser iluminados.
  - Qué se entiende por línea en un dispositivo raster?.
  - Cómo se evalúa el éxito de algún algoritmo propuesto?.

Encontrar el pixel

Casos especiales:

- Línea Horizontal:

  Dibujar el pixel *P* e incrementar la coordenada *x* en un valor de 1 hasta alcanzar el pixel final *Q*.

- Línea Vertical:

  Dibujar el pixel *P* e incrementar la coordenada *y* en un valor de 1 hasta alcanzar el pixel final *Q*.

- Línea Diagonal:

  Dibujar el pixel *P* e incrementar ambas coordenadas *x* e *y* en un valor de 1 hasta alcanzar el valor final *Q*.

¿Cómo sería el caso general?

Estrategia 1: Algoritmo Incremental (1/3)

El Algoritmo Básico

- Encontrar la ecuación de la línea que conecta los dos puntos P y Q.
  $y = mx + B$
  donde m = *pendiente* = $(y_1-y_0) / (x_1-x_0)$, B = a donde y es intersecada
- Comenzando por el punto de más a la izquierda P,
  - incremenart x_i en **1**
  - calcular $y_i = mx_i + B$
- Iluminar el pixel en (x_i, y_i)

  o mas específicamente en $(x_i, \texttt{Round}(y_i))$

  donde $\texttt{Round}(y_i) = \texttt{Floor}(y_i + 0.5)$

Código de Ejemplo

```
// Assumes -1 <= m <= 1, x0 < x1

void Line(int x0, int y0,
          int x1, int y1, int value) {
   int x;
   float y;
   float      dy = y1 - y0;
   float      dx = x1 - x0;
   float      m = dy / dx;

   y = y0;
   for (x = x0; x < x1; x++) {
       WritePixel(x, Round(y), value);
       y = m * x;
   }
}
```

Estrategia 1: Algoritmo Incremental (2/3)

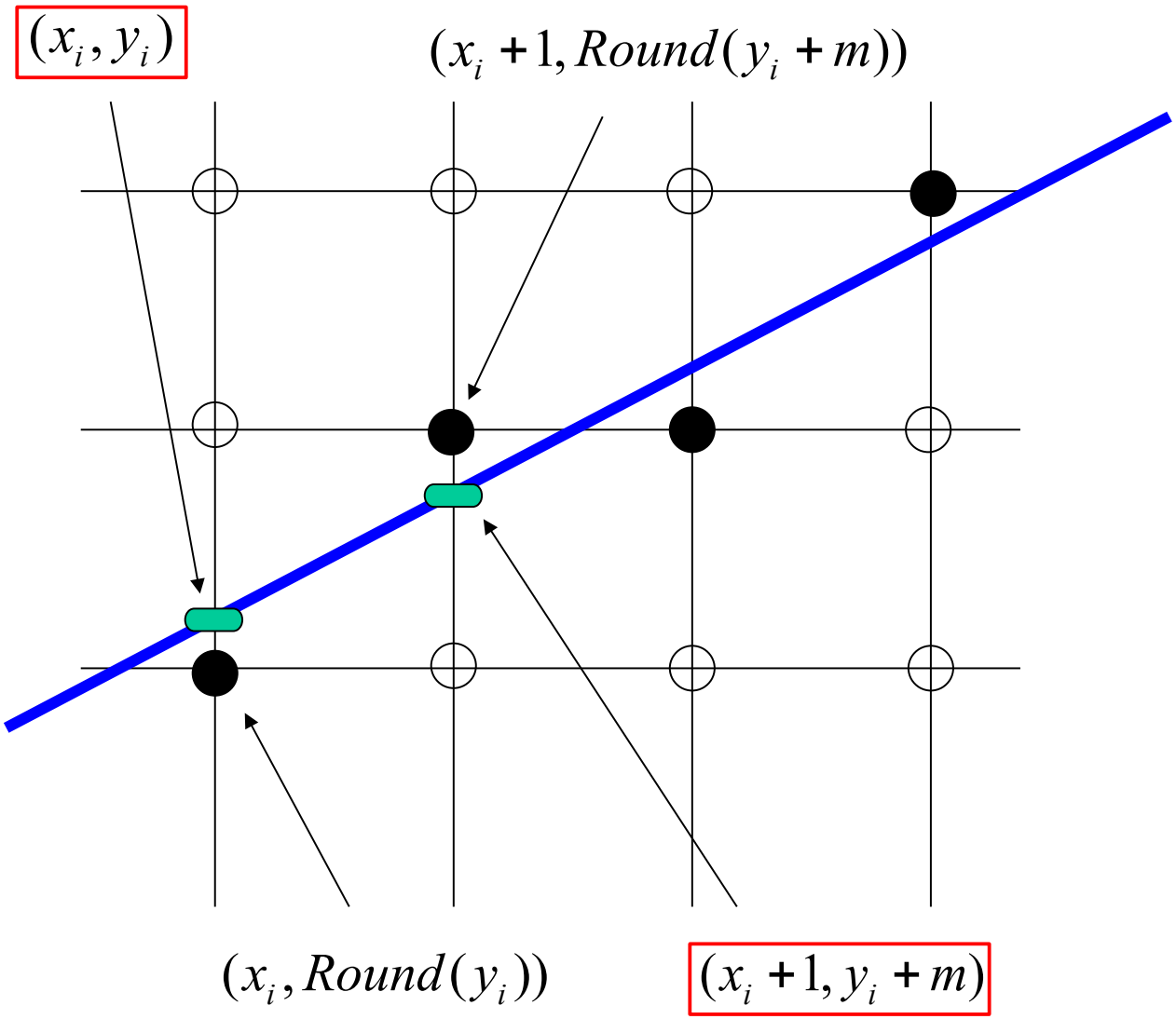
El Algoritmo Incremental:

- Cada interación requiere una operacion flotante que consume tiempo de proceso
  - debe intentar eliminarse.
- $y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = mx_i + m\ x + B = (mx_i + B) + m\,\Delta x = y_i + m\,\Delta x$
- Como $\Delta x = 1$, luego $\boxed{y_{i+1} = y_i + m}$
- En cada paso, para encontrar el valor del próximo y se realizan cálculos incrementales basados en el paso anterior.

Código de Ejemplo

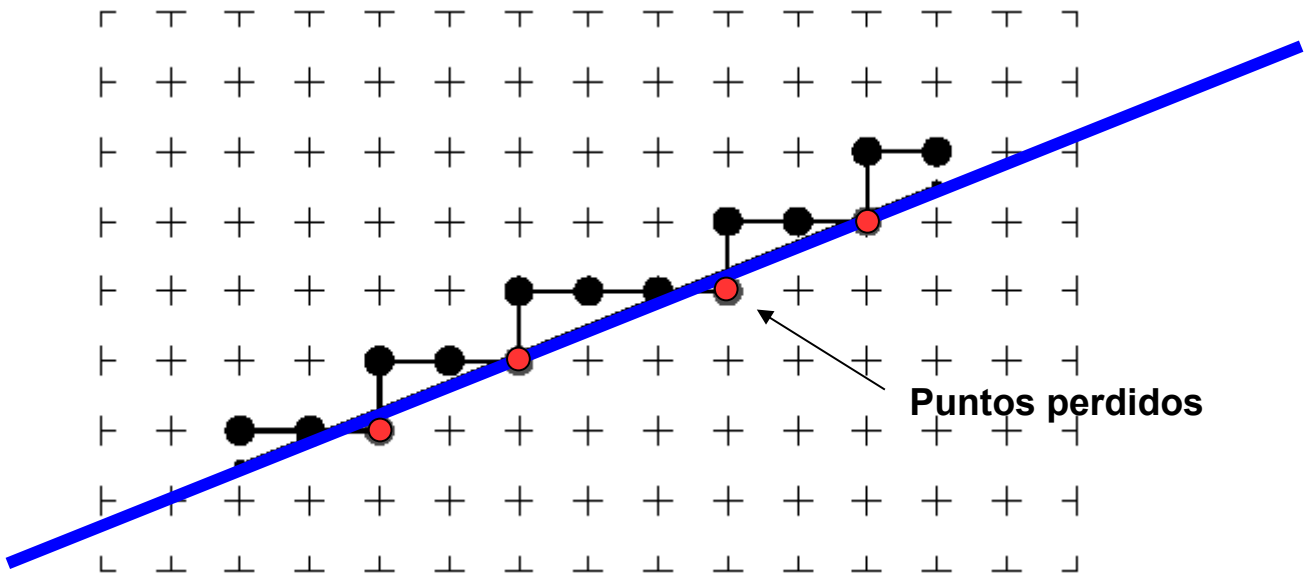
```
// Incremental Line Algorithm
// Digital Differential Analyzer DDA
// Assumes -1 <= m <= 1, x0 < x1

void Line(int x0, int y0,
          int x1, int y1, int value) {
   int x,
   float y;
   float      dy = y1 - y0;
   float      dx = x1 - x0;
   float      m = dy / dx;

   y = y0;
   for (x = x0; x < x1; x++) {
       WritePixel(x, Round(y), value);
       y = y + m;
   }
}
```

Estrategia 1: Algoritmo Incremental (3/3)



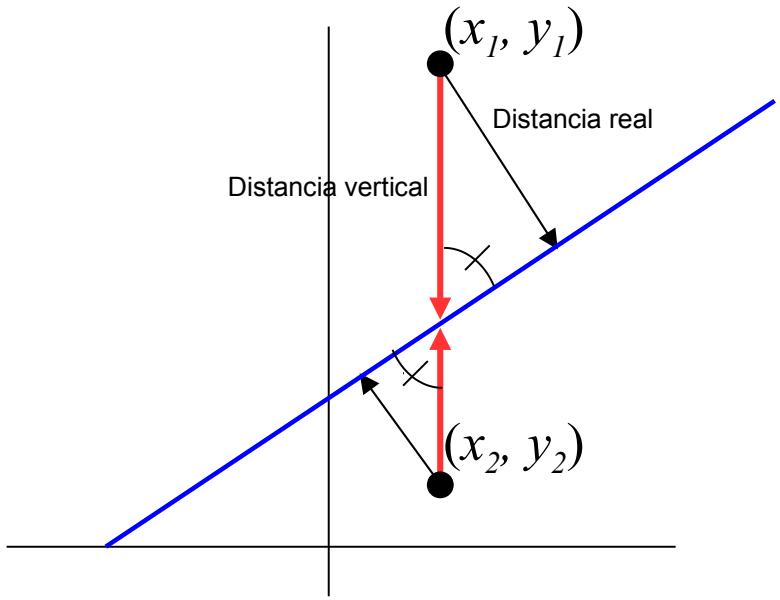
Problema del Algoritmo Incremental

- Las variables ***y*** y ***m*** deben ser reales o fraccionarias pues la pendiente es una fracción.
  - las líneas verticales / horizontales se tratan como caso especial
- La operación de redondeo consume tiempo.
- Dado que las variables tienen precisión limitada, la suma repetida de un ***m*** no entero introduce un error acumulativo en la reconstrucción. El problema se agrava con líneas largas.



Distancia Vertical

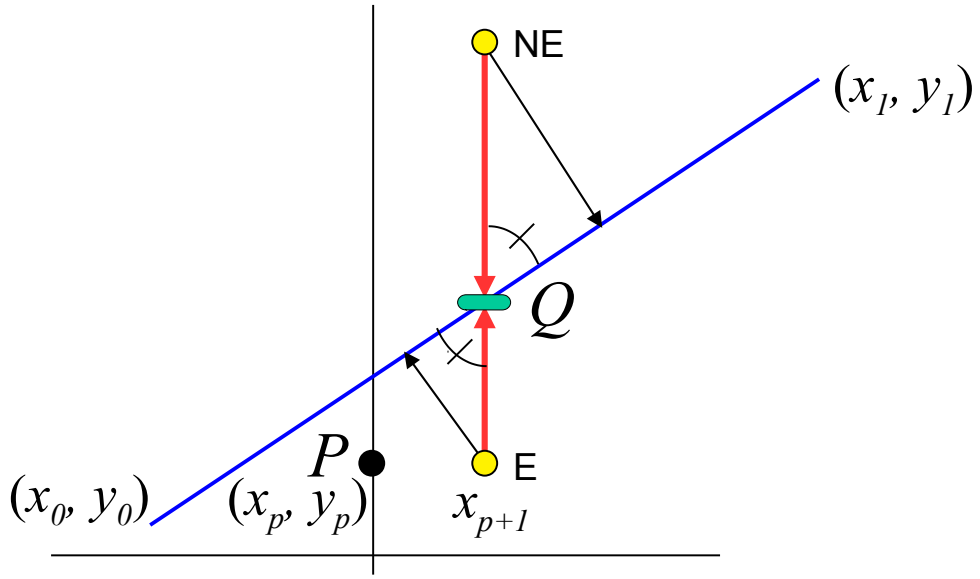
- Se podría utilizar la distancia vertical para establecer que punto se encuentra más cercano?



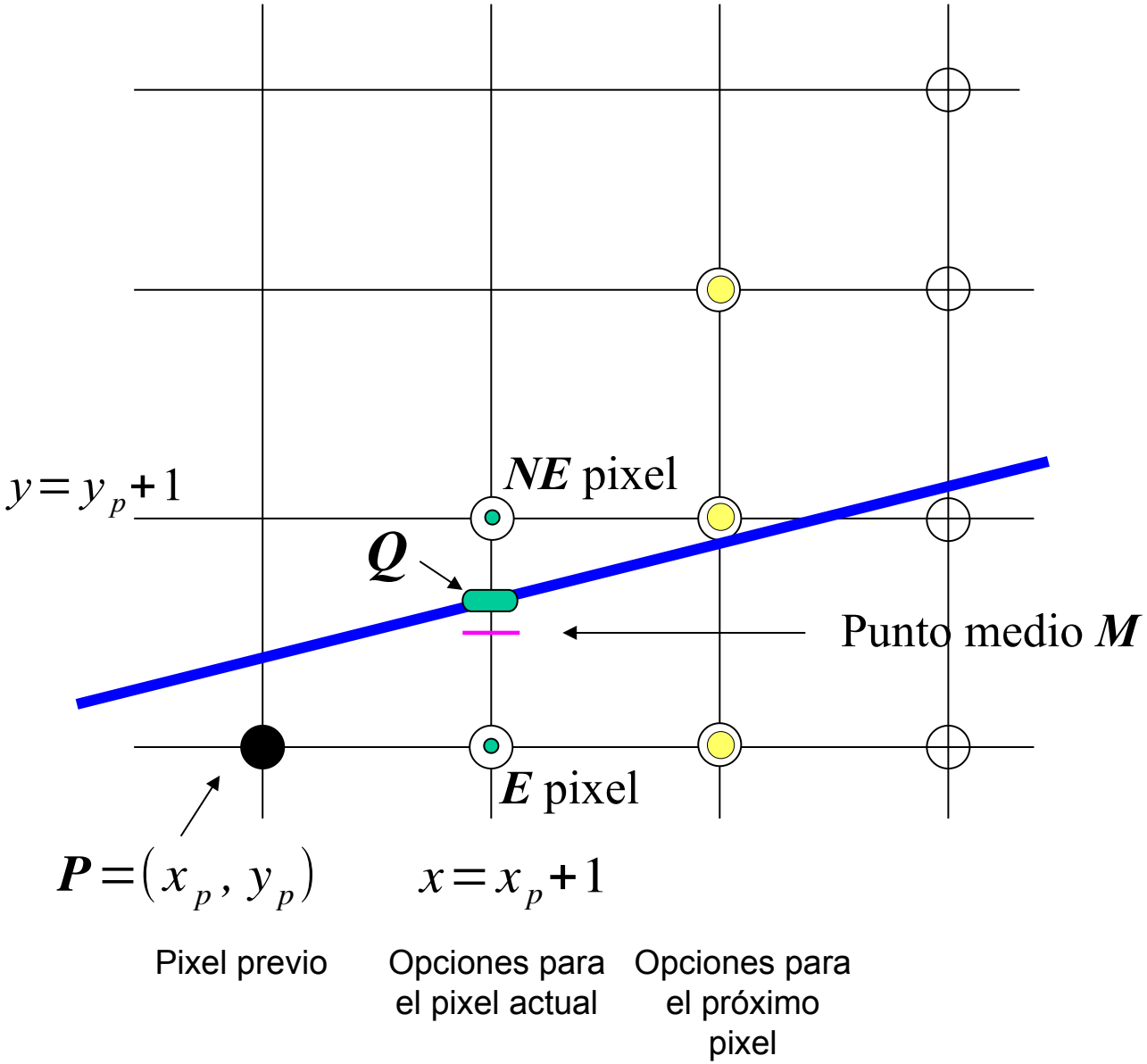
- Por triángulos semejantes, para cada punto la distancia real a la línea proyectada (vector negro) es directamente proporcional a la distancia vertical a la línea (vector rojo).
- Por consiguiente, el punto con menor distancia vertical a la línea proyectada se encuentra más cerca de dicha línea.

Estrategia 2: Algoritmo del Punto Medio (1/3)

- Asuma que la línea tiene pendiente positiva suave.
  (0 < pendiente < 1); variaciones de la misma pueden realizarse mediante el intercambio de los valores de las coordenadas.
- Denominemos al punto inferior izquierdo (x_0, y_0) y al superior derecho (x_1, y_1).
- Asuma que se ha seleccionado el pixel P en (x_p, y_p).
- Se debe elegir el próximo pixel a pintar entre: el que se encuentra a la derecha (pixel **E**) o el que se encuentra a la derecha y hacia arriba (pixel **NE**).
- Donde Q es el punto de intersección de la línea original discretizada con la línea de la grilla en $x = x_p + 1$.



Estrategia 2: Algoritmo del Punto Medio (2/3)



Estrategia 2: Algoritmo del Punto Medio (3/3)

- **Objetivo**: elegir el punto de grilla que se encuentre mas cercano al punto de intersección Q.

- **Problema**: encontrar una forma de calcular automáticamente de que lado de la línea ideal se encuentra el punto medio M de la grilla, y hacerlo con la menor cantidad de cálculos.

  Variable de decisión: observar de que lado de la línea ideal se encuentra el punto medio M de la grilla:
  - E se encuentra cercano a la línea ideal si el punto medio M esta por arriba del punto Q.
  - NE se encuentra cercano a la línea ideal si el punto medio M esta por debajo del punto Q.

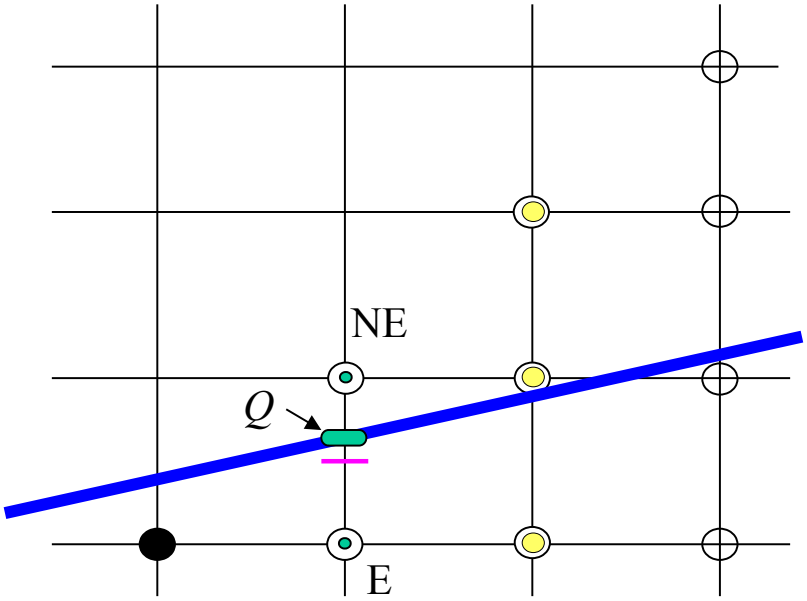
- **Solución**: utilizar la representación implícita del segmento de línea.

  $F(x, y) = a*x + b*y + c = 0$ para coeficientes $a, b, c,$ donde $a, b \neq 0$

  Actualizar en forma incremental la variable de decisión para eliminar cálculos.

- Reducción del error acumulado: la distancia vertical entre el pixel elegido y la línea ideal es siempre <= ½.

Sketch del algoritmo




```
int x = x0;
int y = y0;

Pixel(x, y);
while (x < x1) {
   if (decision_var <= 0) {
       /* move East */
       update decision_var;
   }
   else {
       /* move North East */
       update decision_var;
       y++;
   }
   x++;
   Pixel(x, y);
}
```

Example Code

```
void MidpointLine(int x0, int y0,
                  int x1, int y1, int value)
{
   int  x = x0;
   int  y = y0;

   int  dx = x1 - x0;
   int  dy = y1 - y0;
   int  d = 2 * dy - dx;  // decision variable
   int  incrE = 2 * dy;
   int  incrNE = 2 * (dy - dx);

   writePixel(x, y, value);
   while (x < x1) {
        if (d <= 0) {    // East Case
                d = d + incrE;
        } else {         // North East Case
                d = d + incrNE;
                y++;
        }
        x++;
        writePixel(x, y, value);
   }               /* while */
}                  /* MidpointLine */
```

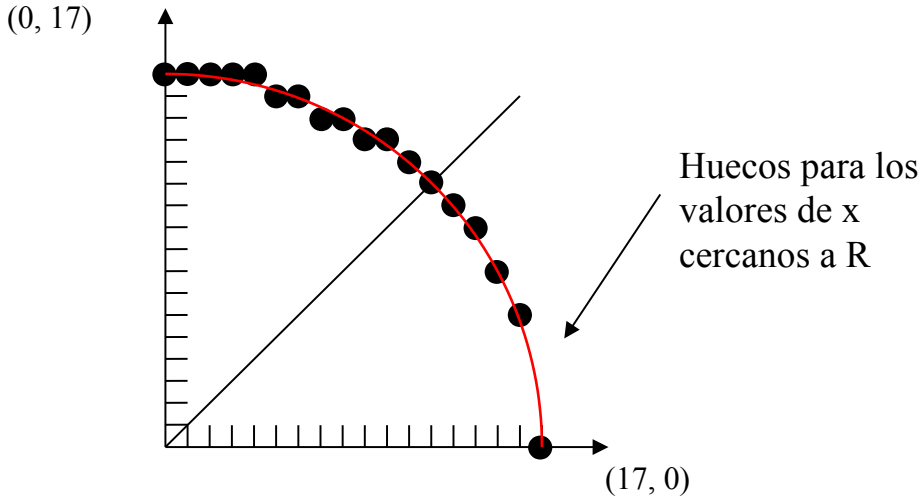
Digitalizar Círculos

- **Versión 1** – a partir de la Función Explícita

$$y = \sqrt{R^2 - x^2}$$

```
For x = -R  to R
   y =  sqrt(R² – x²);
   Pixel (round(x), round(y));
   Pixel (round(x), round(-y));
```

Malo



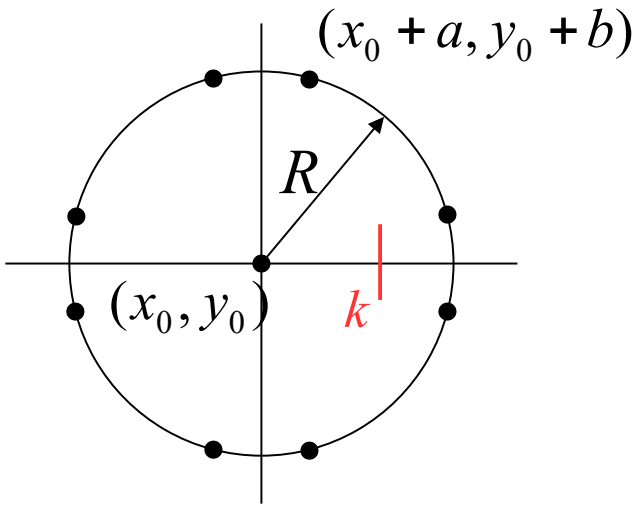
- **Versión 2** – Coordenadas Polares

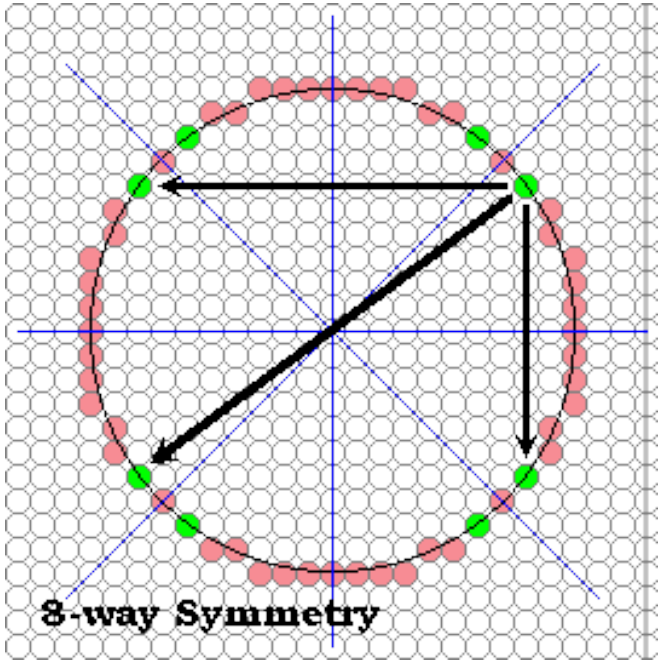
```
For   θ = 0 to 360
   Pixel (round (R • cos(θ)), round(R • sin(θ)));
```

Un poco mejor

Versión 3 — Uso de la Simetría

- **Simetría de 8-lados**





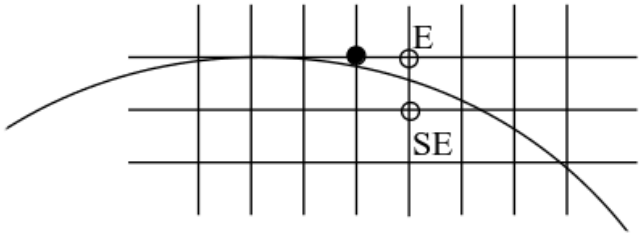
$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

donde $x_0 <= x <= k$ y $k = R/\text{sqrt}(2)$

- Simetría: Si $(x_0 + a, y_0 + b)$ pertenece a la circunferencia, también lo son $(x_0 \pm a, y_0 \pm b)$ y $(x_0 \pm b, y_0 \pm a)$; por consiguiente existe simetría de 8-lados.
- A los fines prácticos, calcular los pixels del primer segmento comenzando en $(x_0, y_0 + R)$.

Consideraciones para el algoritmo

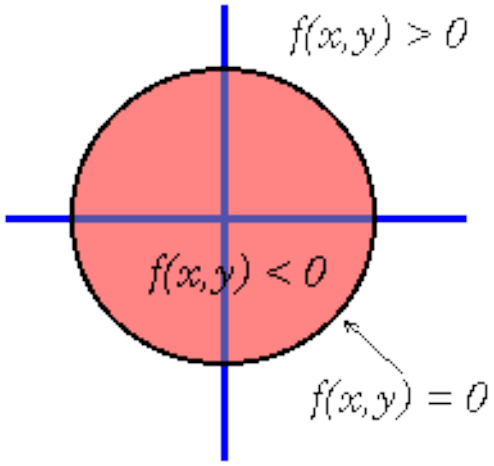
- A partir de la posición $(x_0, y_0 + R)$, avanzar hacia la derecha calculando los pixels a pintar.
- Si se ha evaluado el pixel (x, y), se debe examinar $(x + 1, y)$ y $(x + 1, y - 1)$ para decidir si moverse hacia el **E** o hacia el **SE**



- Es necesario disponer de un parámetro de decisión.
- En forma genérica se puede determinar donde se encuentra ubicado un punto mediante la ecuación implícita de la circunferencia

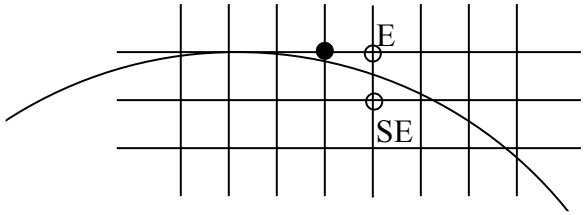
$$F(x,y) = x^2 + y^2 - R^2 = 0$$

Si $F(x,y) = 0$, el punto (x,y) pertenece a la curva,
< 0, el punto (x,y) está dentro del límite de la curva,
> 0, el punto (x,y) está fuera del límite de la curva.



- Se puede definir una variable de decisión utilizando F calculada en el punto medio $= F(midpoint)$

Sketch del algoritmo




```
x = x0;
y = y0 + R;

Pixel(x, y);
while( y > x ) {
   if (decision_var < 0) {
       /* move East */
       update decision_var;
   }
   else {
       /* move South East */
       update decision_var;
       y--;
   }
   x++;
   Pixel(x, y);
}
```

Nota: el proceso se simplifica si se reemplaza x_0 e y_0 con 0, 0

Algoritmo del Punto Medio para Círculos de 8-segmentos

```
MEC (R) /* 1/8th of a circle w/ radius R,
           with center at origin */
{
    int x = 0, y = R;
    int delta_E, delta_SE;
    float decision;
    decision = (x+1)*(x+1) + (y + 0.5)*(y + 0.5) -R*R;
    delta_E = 3;              /* (2*x + 3) */
    delta_SE = -2*R + 5;      /* 2(x-y) + 5 */
    Pixel(x, y);
    while( y > x ) {
        if (decision < 0) {/* Move East */
            decision += delta_E;
            delta_E += 2; delta_SE += 2;
        }
        else {/* Move SE */
            decision += delta_SE;
            delta_E += 2; delta_SE += 4;
            y--;
        }
        x++;
        Pixel(x, y);
    }
}
```

Análisis

- Usa flotantes!
- 1 evaluación, 3 o 4 sumas por pixel
- La inicialización puede ser mejorada
- Si se multiplica por 4, → se mantienen las relaciones..... y se eliminan los flotantes!

Preguntas

- Se están pintando todos los pixels cuya distancia de la línea del círculo real es menor a ½?
- Por qué utilizar “x < y” como criterio de parada del algoritmo?

Otros problemas con la digitalización

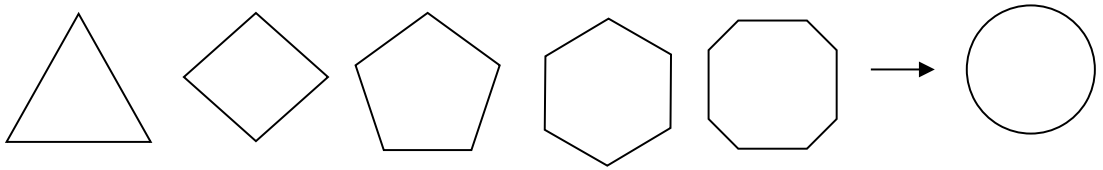
Qué sucede con:

- Elipses?
- Cónicas en general?
- Las primitivas con diseño o dibujo?

Círculo/Elipse - consideraciones

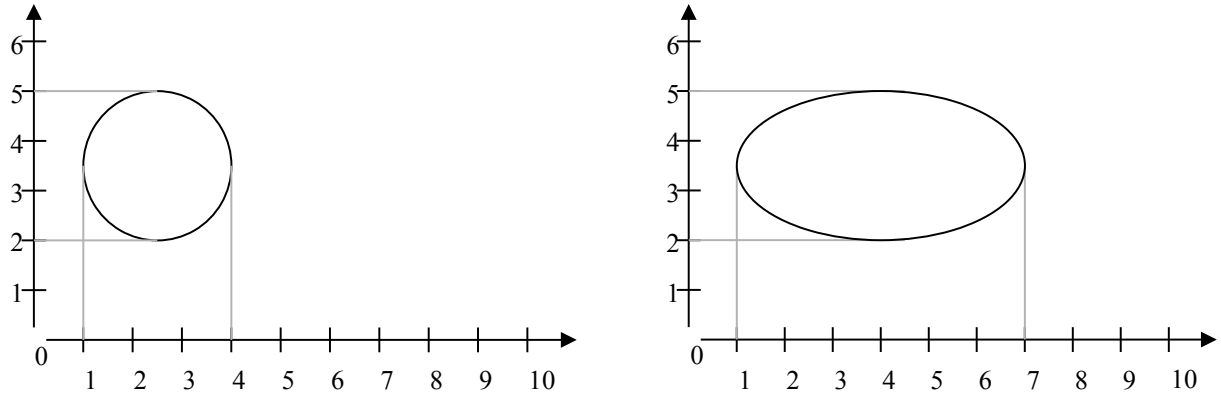
Círculos como polígonos

- Informalmente, un polígono regular con > 15 lados



Ellipses alineadas a los ejes

- Un caso particular de círculo: un círculo escalado en los ejes x o y



Ejemplo: altura en eje *y*, permance 3, mientras que la longitud en eje *x*, cambia de 3 a 6

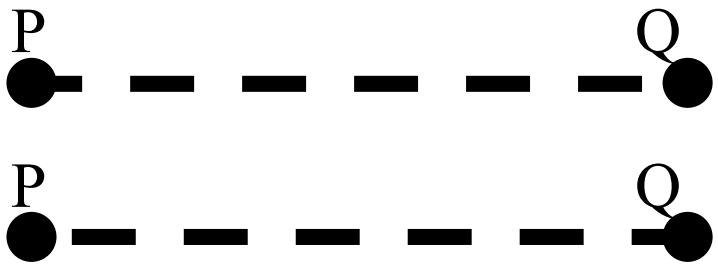
Líneas con Diseño

- Los diseños pueden ser geométricos o cosméticos
  - los cosméticos pueden ser un fondo o un patrón de diseño.

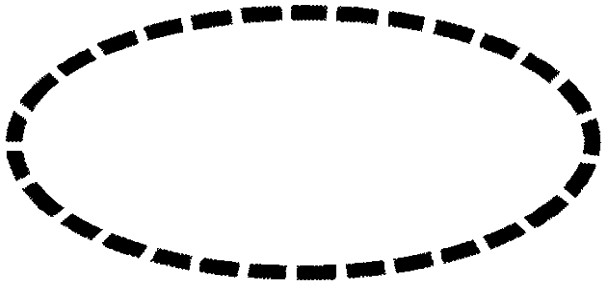
Línea con diseño geométrico

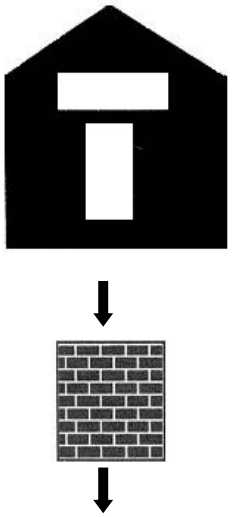
Línea con diseño cosmético

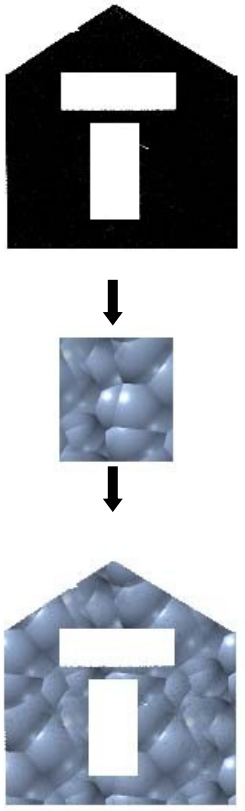
- La línea con diseño geométrico *P* - *Q* no es la misma que la línea *Q* - *P*.



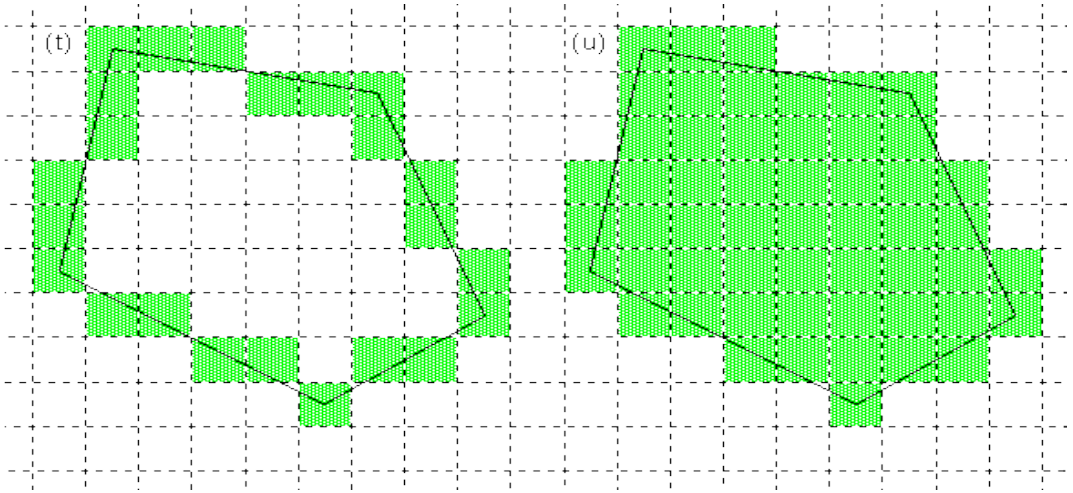
Diseño Geométrico vs. Diseño Cosmético

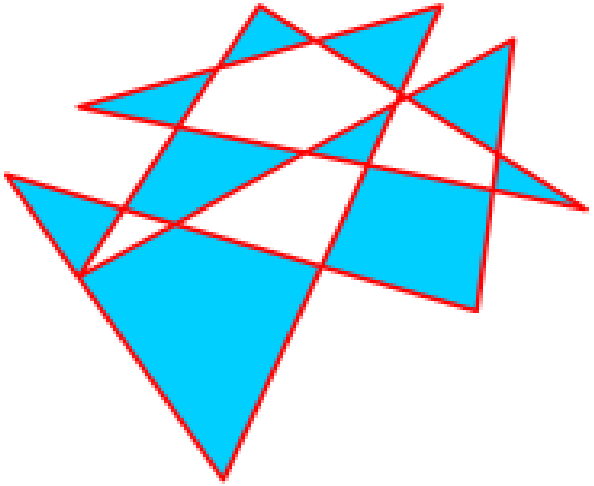






Relleno de Polígonos

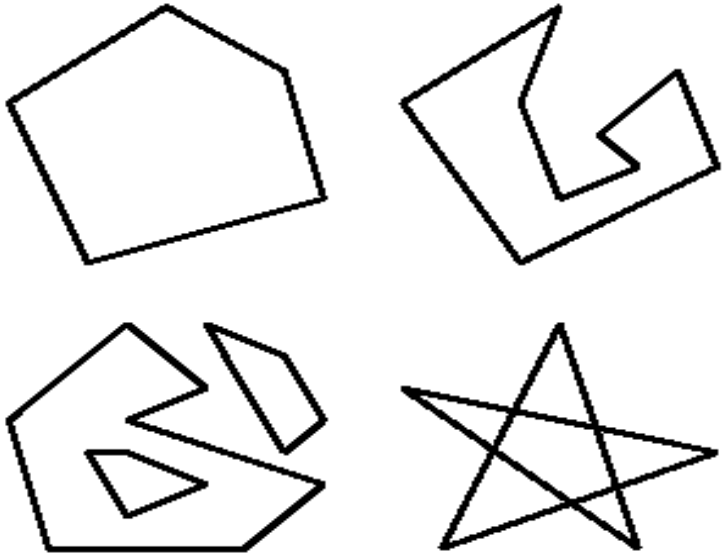




Dpto. de Informática
Fac. Cs. Físico-Mat. y Nat.
Universidad Nacional De San Luis
Argentina

Relleno de Polígonos

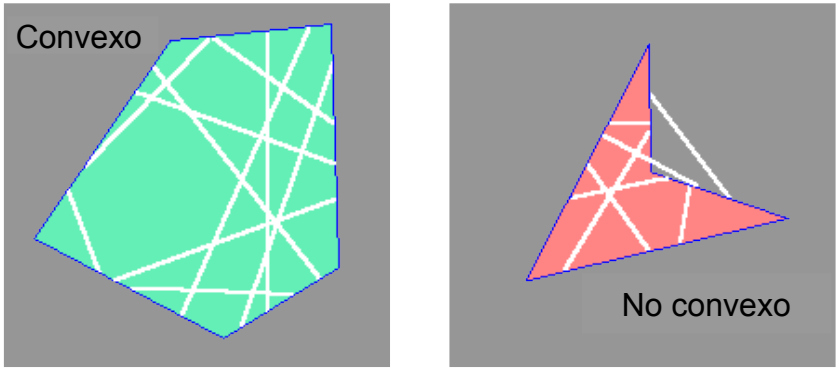
- Se desea hacer mucho más que pintar líneas y curvas.
- En particular, se desea rellenar el interior de cualquier figura.
- La tarea de rellenado se puede dividir en 2 partes:
  - Qué pixel pintar?. Depende de la forma de la figura (convexa o no convexa).
  - Con qué valor rellenar.



Tipos:
- Triángulos
- Trapezoides
- Quadriláteros
- Convexos
- Cóncavos
- Auto-intersecados
- Generales

Polígonos

- Qué significa que un polígono sea convexo?
  - un polígono es convexo si y sólo si cualquier segmento de línea que conecta dos puntos de su contorno se encuentra totalmente contenido en el polígono o en uno de sus lados.



- Figuras convexas: se podría rellenar su interior mediante el algoritmo de pintado de líneas:
  - que pixels?,
    utilizar reiteradamente el algoritmo de pintado de línea para determinar las primitivas de contorno, y posteriormente
  - rellenar los espacios entre las primitivas de izquierda a derecha.

Rellenado de Rectángulos

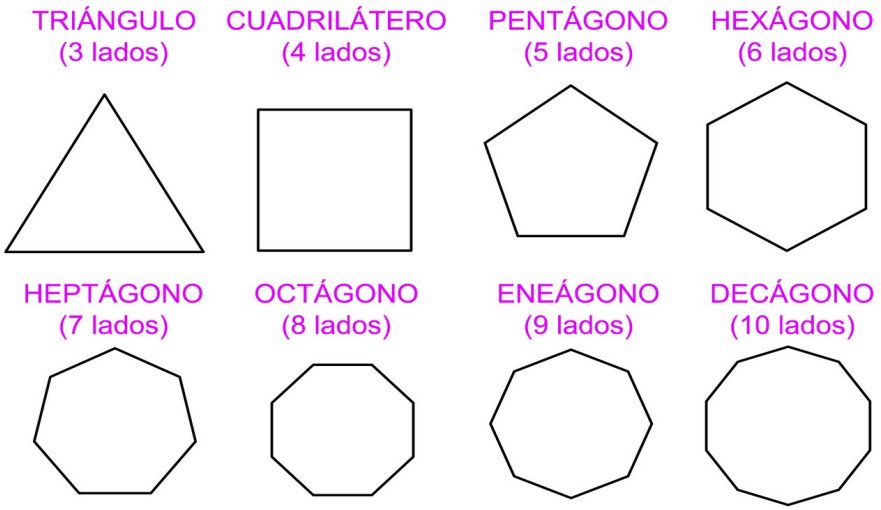
- Para rellenar un rectángulo:
  - pinte cada pixel que se encuentra en la línea de scan desde izquierda a derecha.
- Es una simple anidación de loops - **`for`**

```
for (y from y_min to y_max of the rectangle) // Scan Line
   for (x from x_min to x_max)               // Span
       WritePixel(x, y, value);
```

- Se pueden procesar múltiples pixels en un mismo intervalo con el objeto de:
  - escribir el frame buffer una sola vez.
  - minimizar el tiempo consumido en accesos a memoria.
- Problema: Considerar dos rectángulos que comparten el mismo lado de contorno.
  - si se digitaliza cada rectángulo por vez se pintará el lado compartido 2 veces.

Polígonos Regulares

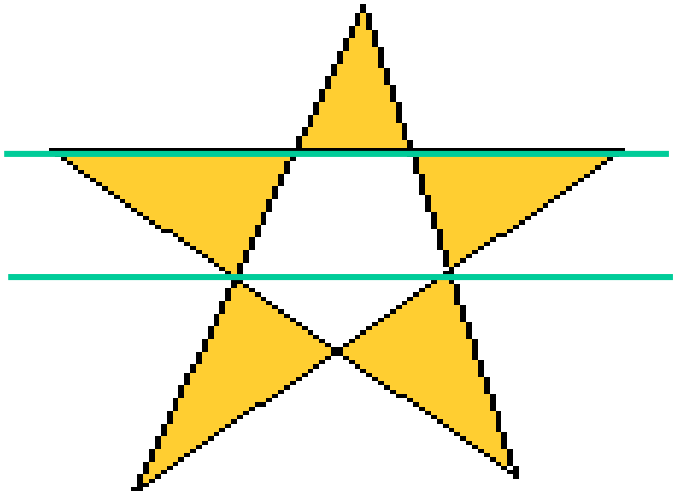
- El primer algoritmo, ***Algoritmo de relleno de Paridad***, muy simple.
  - procesa una fila o línea de scan por vez, de izquierda a derecha.
  - cada línea de scan comienza en un punto fuera de la figura.
  - al encontrar el borde de la figura, debe cambiar su estado de externo a interno a la figura.
  - dentro de la figura, reemplaza el valor del pixel con el color de pintado hasta encontrar el próximo borde.



- Desventajas.
  - trata cada figura en forma aislada.
  - trabaja bien con figuras ideales.
  - no funciona bien con figuras que se auto-intersectan.

Polígonos Regulares

- A pesar de sus problemas, el algoritmo de relleno es muy popular en algunas aplicaciones tales como el pintado de fuentes o polígonos.
  Características:
  - es rápido y se adapta perfectamente a implementar en hardware.
  - la mayoría de los paquetes no permiten figuras que se auto-intersecan.
  - el algoritmo da buenos resultados.



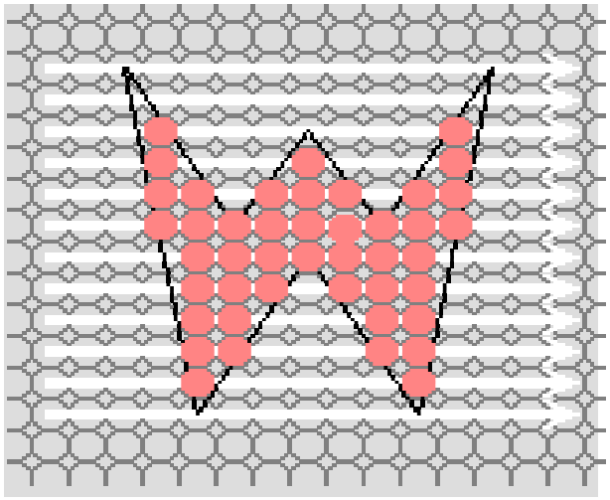
- Generalización? / Optimización?:
  Sería conveniente tomar ventaja de la topología de la figura y más aún, de la escena...

Coherencia

- **Coherencia**: la relación, conexión o unión de unas cosas con otras, o aquello que interconecta o mantiene unidas las partes de un todo.
- Pare el caso, se podría explotar la coherencia considerando solamente aquellos pixels donde se producen cambios.
- Diferentes tipos de coherencia a explotar según la necesidad.
- ***Coherencia de la Escena**:* se hace referencia a las diferentes partes de una escena que podrían brindar una ventaja al momento de procesar el pintado.
- ***Coherencia Espacial**:* las primitivas no cambian de pixel a pixel dentro de un intervalo o de una línea de scan a otra.
- ***Coherencia de Línea de Scan**:* en algunas figuras, tal como los rectángulos las líneas de scan consecutivas son iguales, en algunas figuras generales con líneas iguales se denomina coherencia de Bordes/Lados.

Polígonos Generales

- Un algoritmo de digitalización de un polígono general debe tratar por igual polígonos cóncavos y convexos; aún los que se intersecan y tienen huecos.
- Intentar
  - Trabajar sobre tajadas (intervalos/span) de un polígono de izquierda a derecha entre sus bordes.
  - Los extremos del intervalo sean calculados mediante un algoritmo incremental que calcule la intersección de los bordes con la línea de scan.



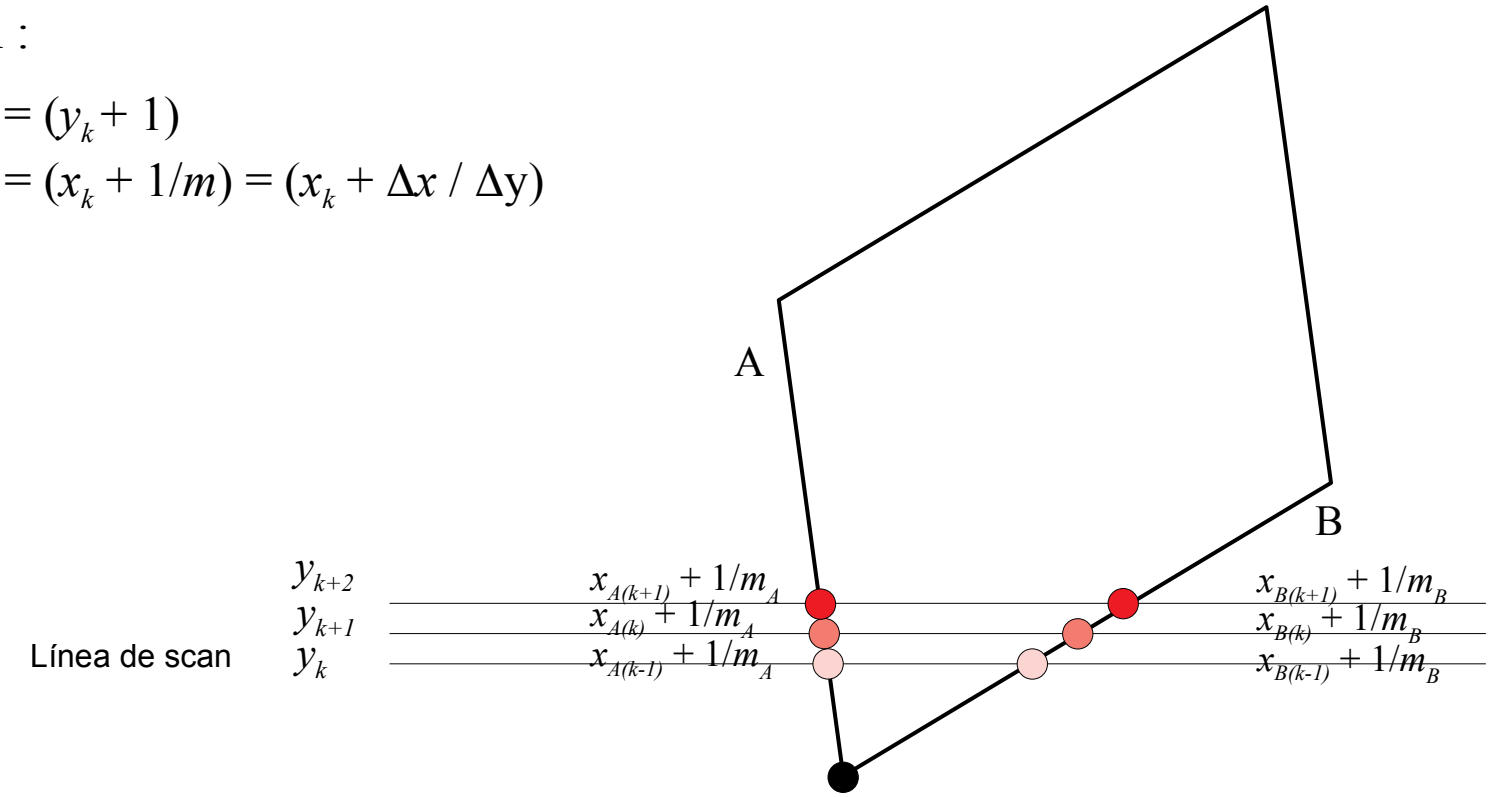
- Numerosas soluciones. Estos algoritmos pueden ser llamados algoritmos coherentes dado que una parte del cómputo proviene de pasos anteriores.

Explotar la coherencia

Paso 1: Intersección de los bordes del polígono con la línea de scan -
Utilizar la coherencia de la reconstrucción de bordes en el cálculo de las intersecciones con la línea de scan corriente (Coherencia de bordes).

- Si la intersección del segmento del polígono con la línea de scan previa es conocida, se puede utilizar un algoritmo de dibujo de línea para calcular la intersección actual.
- Utilizar el algoritmo de tipo incremental donde $\boldsymbol{y = mx}$, y a partir de la intersección previa (x_k, y_k) con :

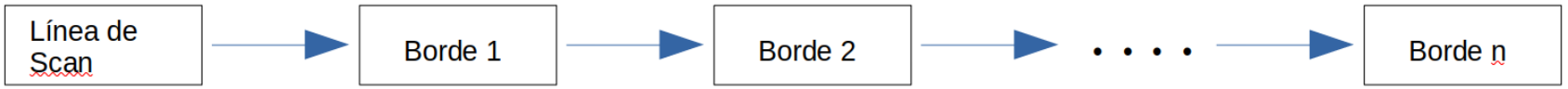
$$y_{k+1} = (y_k + 1)$$
$$x_{k+1} = (x_k + 1/m) = (x_k + \Delta x / \Delta y)$$

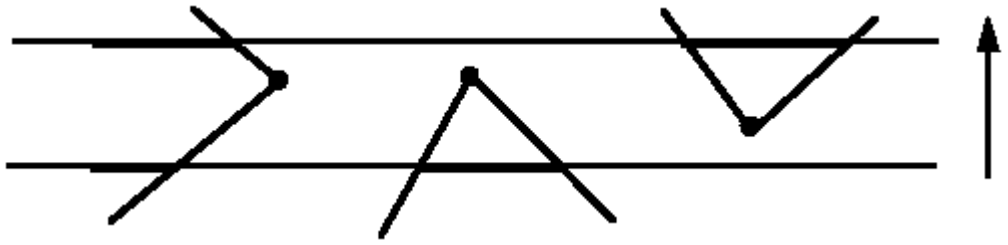


Explotar la coherencia

Paso 2: Bordes en una misma línea de scan -
Utilizar la coherencia de orden (Coherencia de bordes – Coherencia de línea de scan).

- En cada Línea de scan **y,** generar una lista de intersecciones ordenadas por **x** de manera de poder determinar los intervalos.
- Toda vez que en la línea de scan un nuevo borde comienza, se debe insertar en la lista de intersecciones ordenadamente.
- Si ningún lado comienza en la línea de scan actual, el orden de los comienzos de segmentos permanecen igual.





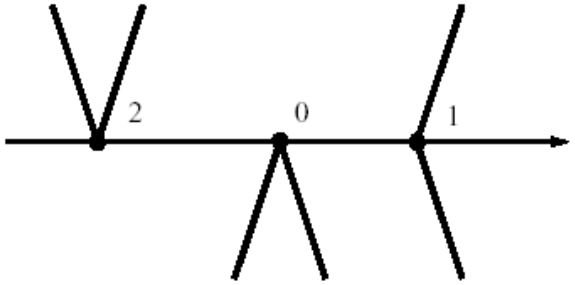
reemplazar suprimir insertar

Explotar la coherencia

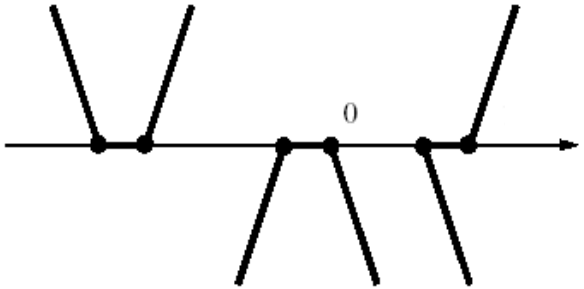
Paso 3: Pintar los intervalos -
Considerar los casos especiales de vértice del polígono en la línea de scan.
(Coherencia de poĺigonos . Coherencia de línea de scan)

- Para polígonos cerrados, los vértices que pertenecen a dos bordes se encuentran repetidos.

- El uso de la siguiente regla evita problemas:
  - Si el vértice es el vértice inferior del borde, se lo cuenta.
  - en otro caso, se lo ignora.

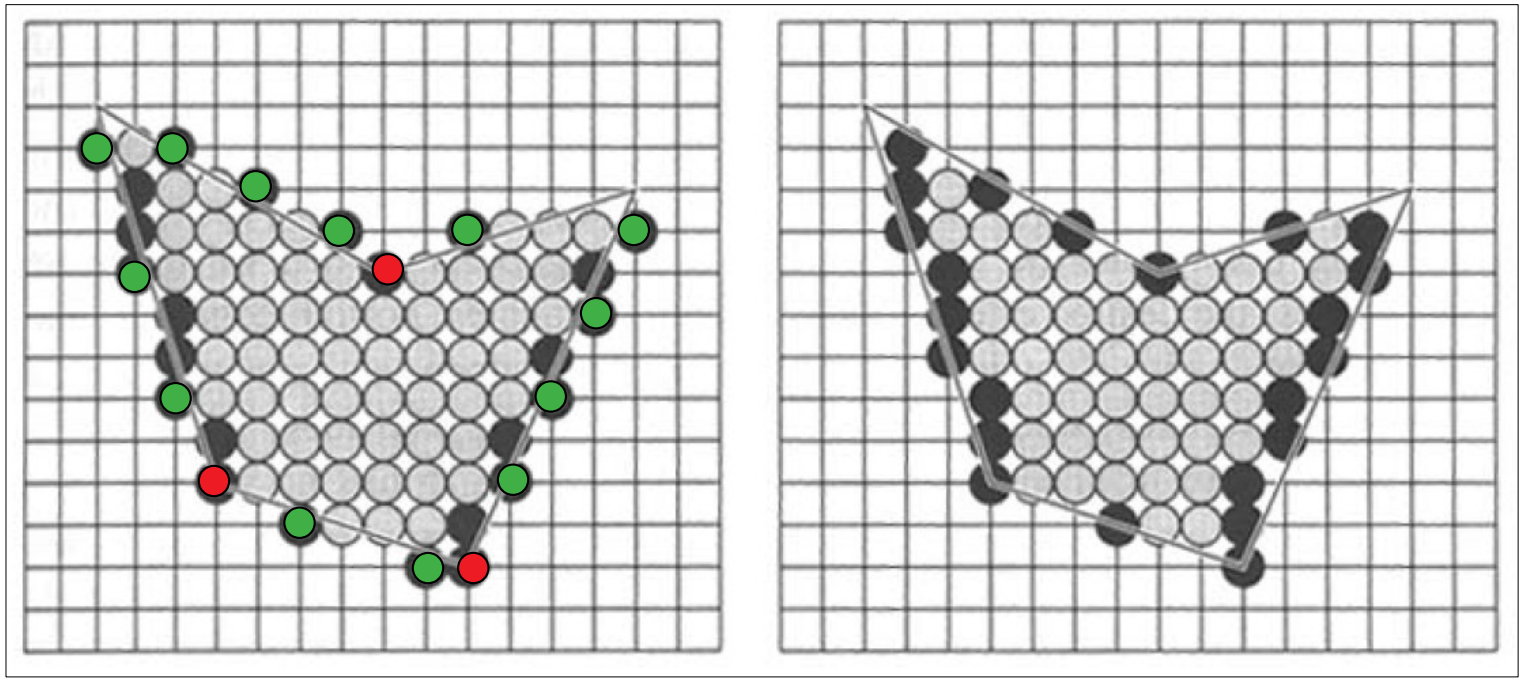


- Qué sucede con los bordes horizontales de un polígono?
  - No se cuentan vértices en bordes horizontales.



Singularidades

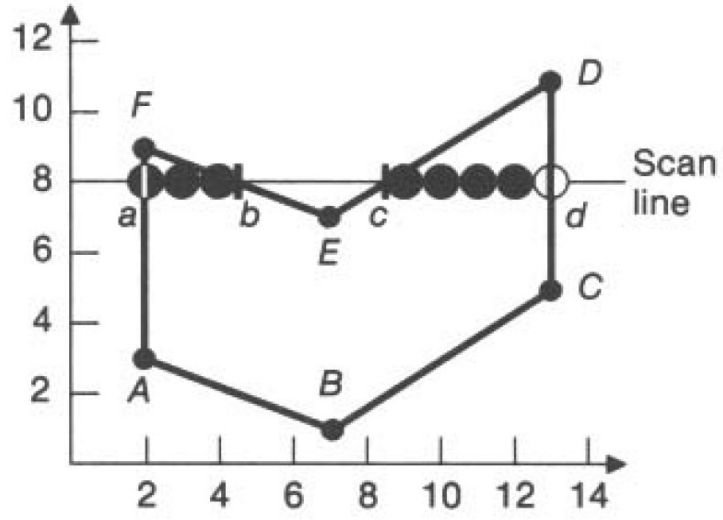
- La estrategia para la determinación de la intersección de bordes en la línea de scan debe ser óptima. No se desea perder ni pixels, ni tiempo.
- Es conveniente dibujar estrictamente aquellos pixels que se encuentren dentro del polígono. No pintar pixels vecinos..



- Se deben considerar los pixels que se repiten por pertenecer a dos vértices.

Algoritmo de Línea de Scan

- Toma ventaja de la coherencia de bordes y en cada línea de scan recuerda el conjunto de bordes que se intersecan con la línea de scan y sus correspondientes puntos de intersección. Permite polígonos auto-intersecados.
- Utiliza una manera simple de determinar los extremos de la línea de scan mediante el algoritmo del punto medio sobre los bordes y mantiene una tabla de extremos de intervalos (span) para cada línea de scan.
- Para cada línea de scan se deben computar tres pasos:
  1) Calcular todas las intersecciones de los lados del polígono con la línea de scan corriente.
  2) Ordenar las intersecciones en orden creciente de coordenada x.
  3) Utilizar una regla de paridad par-impar para calcular los intervalos de pixels que deben ser pintados
     - Comenzar con paridad par.
     - En cada intersección cambiar la paridad.
     - Si la paridad es impar, pintar cada pixel hasta una nueva intersección (esto es un intervalo - span).
- El algortimo de paridad tiene inconsistencias. Recordar reglas.

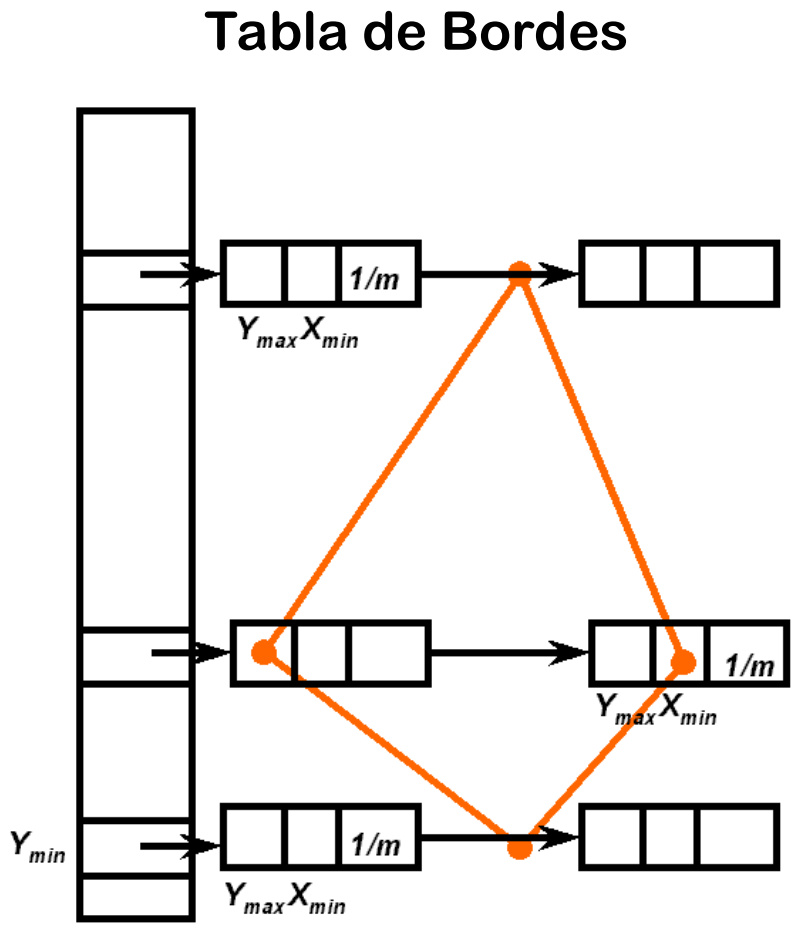


Algoritmo de Línea de Scan

Estructuras soporte:

1) **Tabla Global de Bordes** (*Edge Table* - ET) conteniendo todos los bordes ordenados por coordenada y_{min} como lista de buckets.

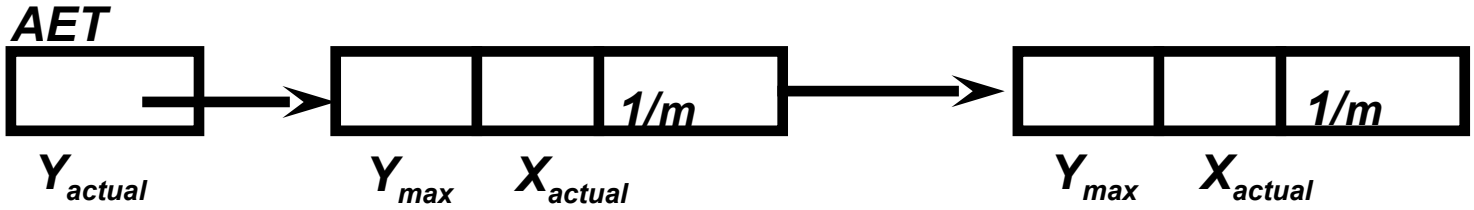
   - En cada lista de buckets, los bordes se ordenan acorde con coordenada x_{min} creciente.
   - Cada bucket contiene coordenada y_{max} del lado, la coordenada x_{min} (el punto final del borde), y el incremento para x utilizado en el paso de una línea de scan a la otra.

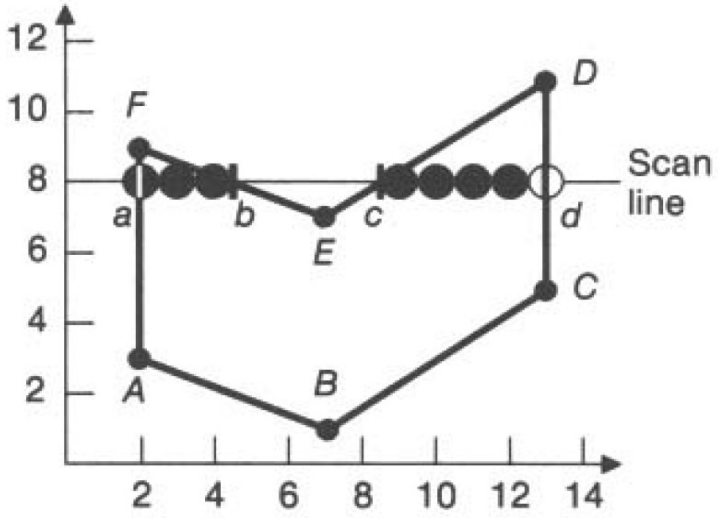


Algoritmo de Línea de Scan

2) **Tabla de Bordes Activos** (*Active Edge Table* - AET) conteniendo el conjunto de bordes activos que la línea de scan interseca.

- Los bordes son ordenados acorde con su intersección con la coordenada ***x*** para luego llenar las tajadas.
  El extremo de una tajada se define por los pares de valores de intersección.





Algoritmo de Línea de Scan

Luego que la Tabla Global de Bordes ha sido creada, los pasos de procesamiento para el algoritmo de línea de Scan son:

- Inicializar y con el menor valor de coordenada que tiene entrada en la tabla de bordes.
- Inicializar la tabla de bordes activos (*Active Edge Table*) en cero.
- Repetir hasta que la AET y ET se encuentren vacias:
  - Mover del bucket de la ET a la AET aquellos bordes cuyos $y_{min} = y_{actual}$
  - Remover de la AET aquellas entradas con $y_{actual} = y_{max}$ (no involucrados en la línea de scan actual), luego ordenar la AET según x.
  - Pintar los pixels de la línea de scan y_{actual} utilizando los pares de coordenadas x de la tabla de AET.
  - Incrementar y_{actual} en 1 (próxima línea de scan).
  - Para cada borde no vertical remanente en la AET, actualizar x para el nuevo y.

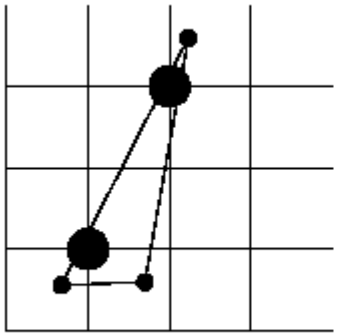
Casos especiales

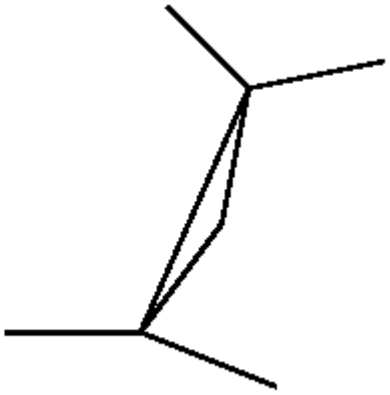
Relleno de Círculos y Elipses.

- Tienen solamente dos extremos de tajada, de modo que se puede utilizar el algoritmo general de polígonos, o utilizar el código para curvas, tomando ventaja de la simetría en el scan debido al arco. Se debe tener cuidado con los cambios de región.
- Se elimina la tabla ET, y solo se conserva la AET.

Astillas

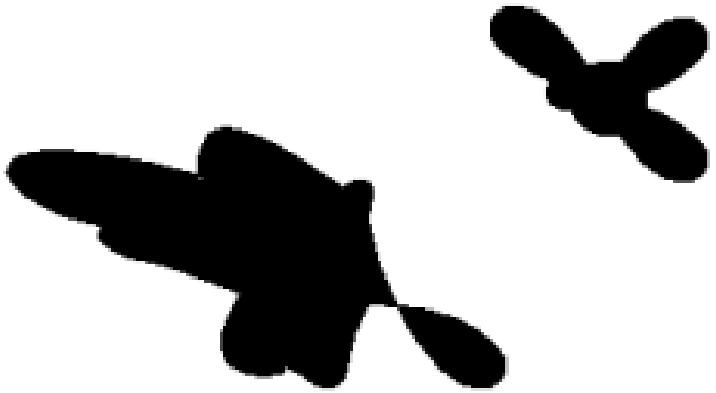
- Polígonos con bordes que se encuentran muy cercanos entre sí. El área a pintar es muy pequeña de modo que su interior no contiene una tajada diferente en la línea de scan.
- Se pueden perder puntos. Se deben utilizar técnicas de *antialiasing*.





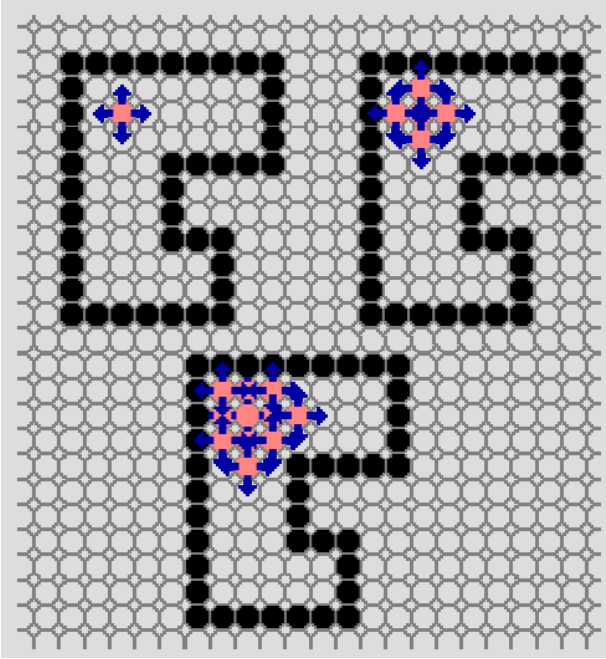
Rellenado de Área

- El algoritmo intenta rellenar figuras con bordes curvos, no polígonos.
- Usualmente, estos algoritmos son utilizados por artistas gráficos los cuales bosquejan los bordes de una figura y luego seleccionan el color o patrón de relleno desde un menú.
- Comienza con un punto conocido dentro de la figura y luego procede a rellenar la figura hacia el exterior del punto.
- En paquetes de pintado interactivo, son funciones de pintado como balde de pintura.



Rellenado de Área

- Áreas definidas por un borde.
- Algoritmo de **Rellenado de Contornos** (*Boundary-Fill algorithm)*. Argumentos: las coordenadas de un punto de comienzo, un color de relleno y un color de contorno.
- Lógica simple. Si no estamos en un borde o punto ya pintado, se debe pintar y luego procesar los vecinos.
- Trata las posiciones vecinas (denominadas 4-regiones conectadas) para determinar si ellas son las frontera de la figura. Si no lo son, se deben pintar y los nuevos vecinos deben ser evaluados.



Rellenado de Área

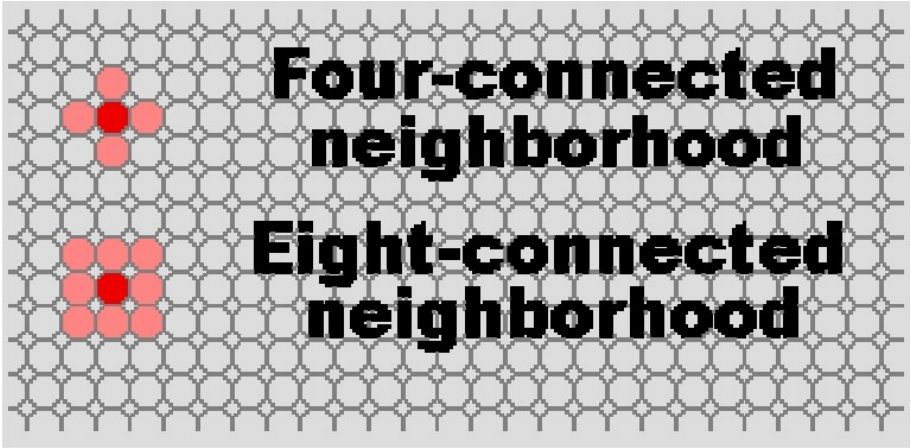
Código de ejemplo para Cuatro Regiones conectadas:

```
void boundaryFill(int x, int y, int fill, int boundary) {
   if ((x < 0) || (x >= raster.width)) return;
   if ((y < 0) || (y >= raster.height)) return;
   int current = raster.getPixel(x, y);
   if ((current != boundary) & (current != fill))
   {
        raster.setPixel(fill, x, y);
        boundaryFill(x+1, y, fill, boundary);
        boundaryFill(x, y+1, fill, boundary);
        boundaryFill(x-1, y, fill, boundary);
        boundaryFill(x, y-1, fill, boundary);
   }
}
```

- Nota: Es una rutina recursiva. Cada invocación de **`boundaryFill()`** se llama a sí misma cuatro veces.

Rellenado de Área

- El algoritmo de relleno depende del tamaño y la conectividad de los vecinos del pixel evaluado.
- Considerar los 8-conectados puede llevar a nudos y grietas, lo que no sucede con un algoritmo de 4-conectados.



Rellenado de Área

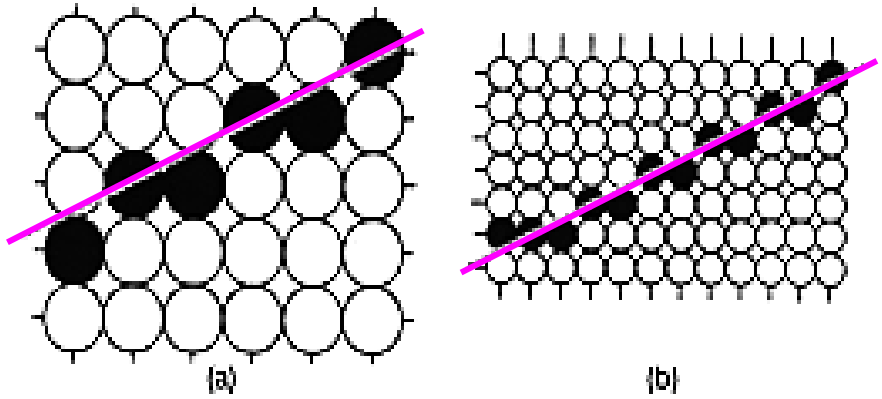
- Áreas no definidas por un borde.
  Se desea reemplazar todos los pixels de un color conectados en un área por un color de relleno.
- El algoritmo de **Relleno de Flujo** (*Flood-fill algorithm*).

```
void floodFill(int x, int y, int fill, int old) {
   if ((x < 0) || (x >= raster.width)) return;
   if ((y < 0) || (y >= raster.height)) return;
   if (raster.getPixel(x, y) == old) {
        raster.setPixel(fill, x, y);
        floodFill(x+1, y, fill, old);
        floodFill(x, y+1, fill, old);
        floodFill(x-1, y, fill, old);
        floodFill(x, y-1, fill, old);
   }
}
```

Escalones – Errores de Muestreo

(Aliasing - submuestreo)

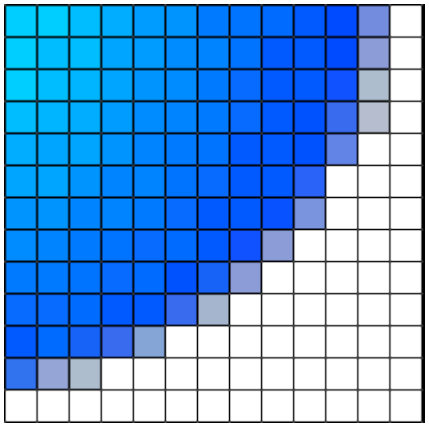
- Representar una línea con valores discretos de pixel significa muestrear una función continua.
- Los escalones son la manifestación de errores de muestreo y pérdida de información.
- Aumentar la resolución de muestreo en x e y reduce el problema.

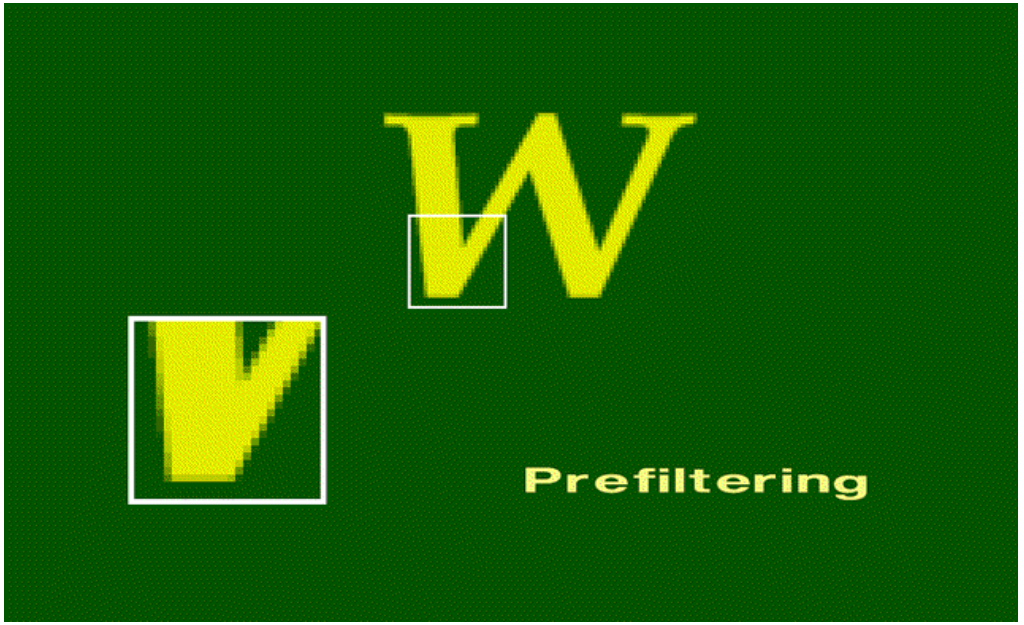


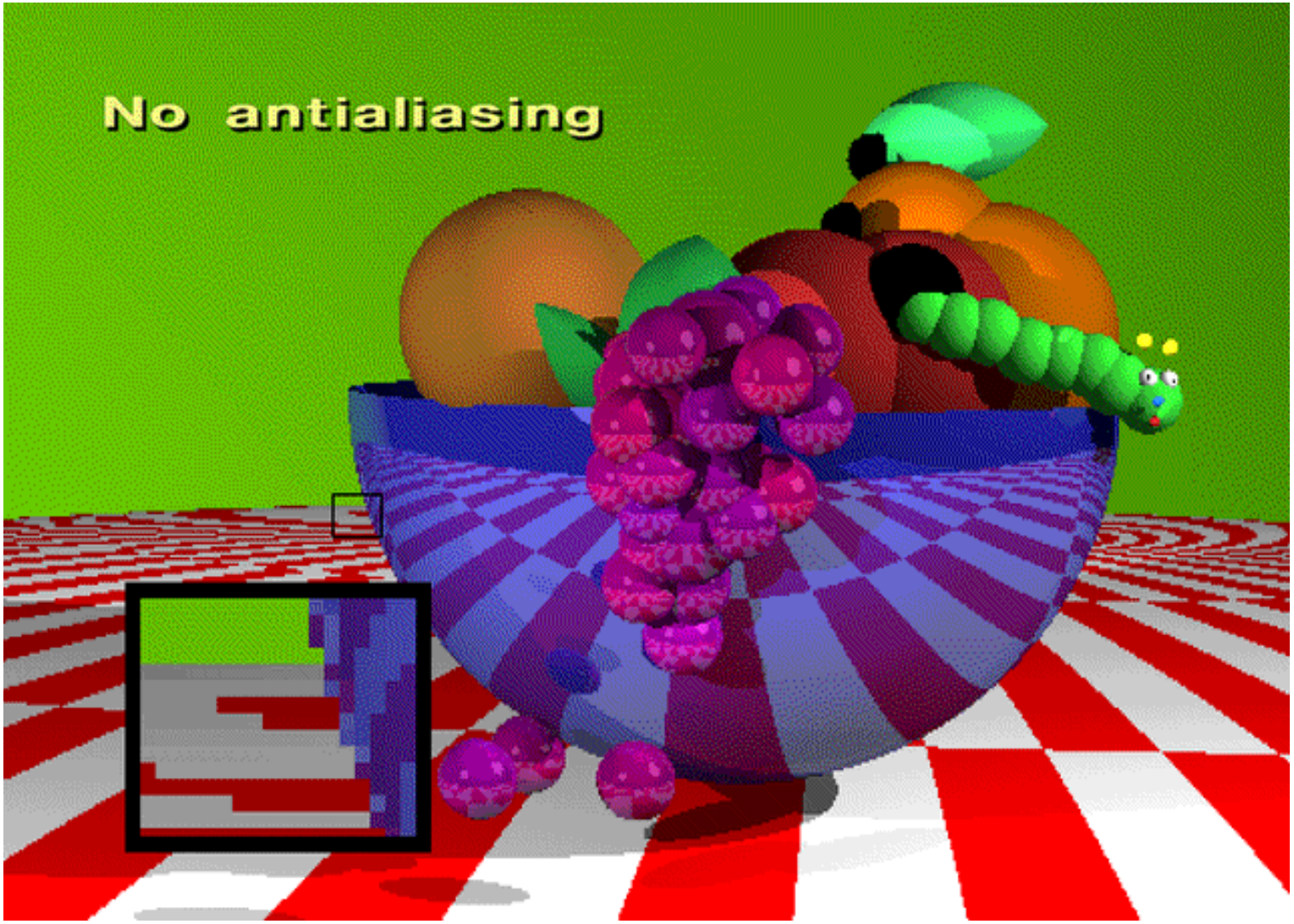
(a) Algoritmo standar del punto medio . (b) La misma línea en un dispositivo con el doble de resolución

- Costo de tiempo de memoria, bandwith de memoria, y tiempo de scanning!
- Las técnicas de suavizado (*antialiasing*) tratan de corregir el problema: Pre-filtering, Supersampling or Postfiltering.

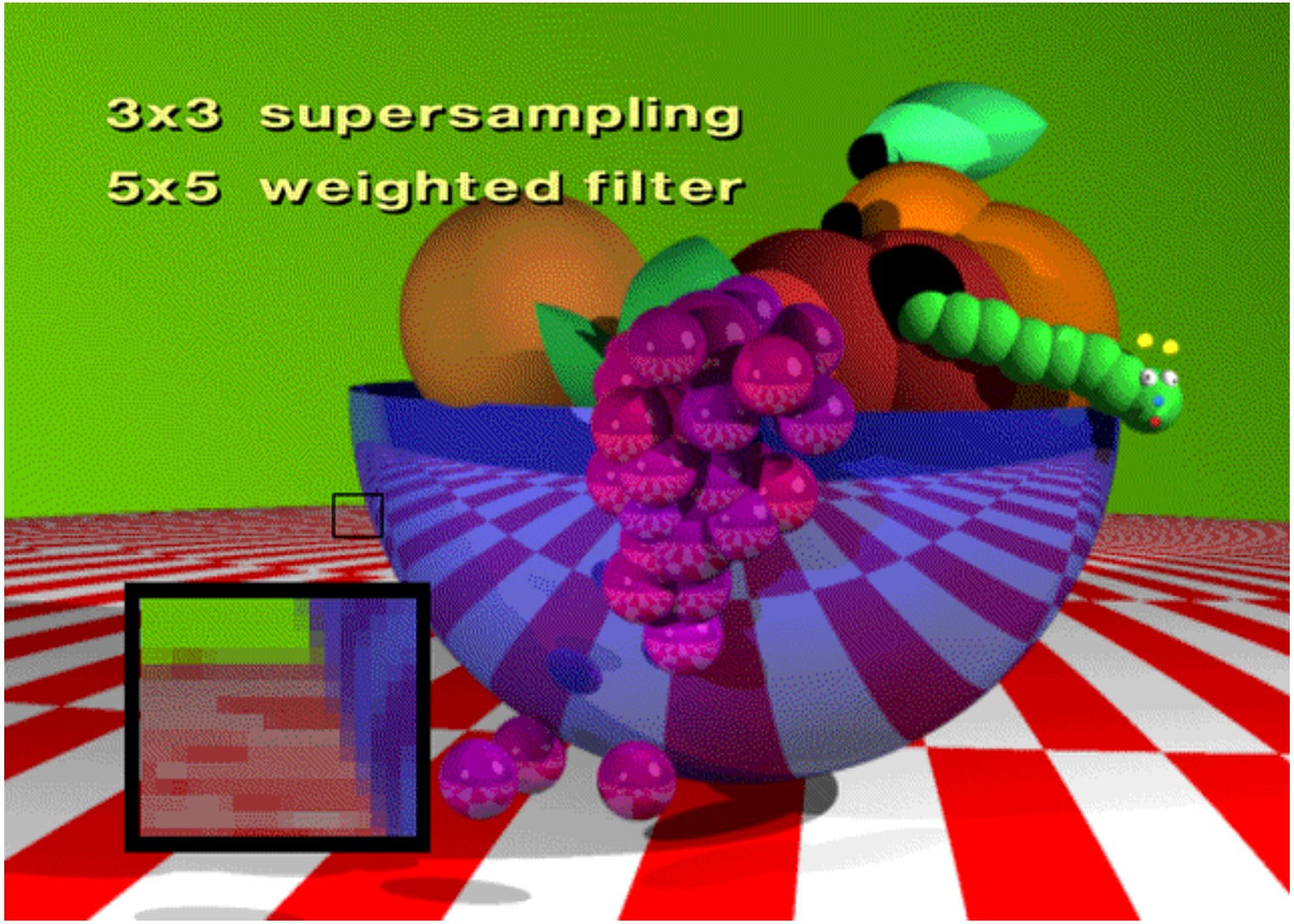

No antialiasing




Prefiltering


No antialiasing


3x3 supersampling
3x3 unweighted filter


3x3 supersampling
5x5 weighted filter